

Error-tolerant Exemplar Queries on RDF Graphs

Zhaoyang Shao
University of Alberta
Edmonton, AB, Canada
zhaoyang@ualberta.ca

Davood Rafiei
University of Alberta
Edmonton, AB, Canada
davood@ualberta.ca

Themis Palpanas
Paris Descartes University
75006 Paris, France
themis@mi.parisdescartes.fr

ABSTRACT

Edge-labeled graphs are widely used to describe relationships between entities in a database. We study a class of queries, referred to as exemplar queries, on edge-labeled graphs where each query gives an example of what the user is searching for. Given an exemplar query, we study the problem of efficiently searching for similar subgraphs in a large data graph, where the similarity is defined in terms of the well-known graph edit distance. We call these queries *error-tolerant exemplar queries* since matches are allowed despite small variations in the graph structure and the labels. The problem in its general case is computationally intractable but efficient solutions are reachable for labeled graphs under well-behaved distribution of the labels, commonly found in knowledge graphs and RDF databases. In this paper, we propose two efficient exact algorithms, based on a filtering-and-verification framework, for finding subgraphs in a large data graph that are isomorphic to a query graph under some edit operations. Our filtering scheme, which uses the neighbourhood structure around a node and the presence or absence of paths, significantly reduces the number of candidates that are passed to the verification stage. We analyze the costs of our algorithms and the conditions under which one algorithm is expected to outperform the other. Our cost analysis identifies some of the variables that affect the cost, including the number and the selectivity of the edge labels in the query and the degree of nodes in the data graph, and characterizes the relationships. We empirically evaluate the effectiveness of our filtering schemes and queries, the efficiency of our algorithms and the reliability of our cost models on real datasets.

1. INTRODUCTION

Graphs are widely used to model relationships, for example between chemical compounds and organisms, objects and scenes in images, entities in RDF, functions and subroutines in a piece of software, etc. An important problem that arises in many of these domains is finding graph structures

that are similar to a query graph.

Searching for similar rather than exact matchings of a query is more desirable when data is noisy or inconsistencies are allowed. For example, in computational biology, the data can be highly noisy because of possible errors in data collection, different thresholds used in experiments and often the difficulty in cleaning the data. Despite the noise, searching for similar biological structures may enable a biologist to learn more about a new organism[6]. In molecular chemistry, identifying similar molecular structures of a target molecule may enable a chemist to design new molecular structures[1]. In social network analysis, searching for similar subgraph may help to identify communities and to predict the network dynamics[15].

In all aforementioned scenarios, one needs to identify the induced subgraphs in a data graph that are similar to a query graph. A number of similarity measures have been proposed[7, 4, 13], of which the graph edit distance[8] is the most general and widely accepted similarity measure. Graph edit distance is defined by the number of edit operations (i.e. the deletion, insertion and substitution of nodes or edges) that is needed to transform one graph into another. A valuable feature of graph edit distance is its error tolerance, allowing user information needs to be captured in the presence of noise and distortion. This paper uses the graph edit distance as its similarity measure between graphs.

There is a large body of work on subgraph similarity search. TALE[16], which indexes each node of the data graph with the node neighbourhood information (such as adjacent node labels, degrees, etc.), allows matching a subset of query nodes before progressively extending these matches. SAPPER[19] takes advantage of pre-generated random spanning trees and a carefully designed graph enumeration order to find approximate subgraph matches. However, these techniques are not applicable in many scenarios. For example, consider searching for isomorphic matchings of the query “Ruby influenced Swift” in the Freebase knowledge graph[3]. This search in TALE will return the query itself. The same query in SAPPER will return the nodes “Ruby” and “Swift”, providing the information that users already know about. The Exemplar queries of Motin et al.[12] proposes a query paradigm, where a query is an example of the expected query answer. More formally, an exemplar query returns all isomorphic matchings where the matching edge of a query has the same label as the query. For instance, the exemplar query “Ruby influenced Swift” (as shown in Figure 1) returns all matches of the form “A influenced B”, e.g. “D influenced Swift”, “Java influenced Closure”, etc.

ACM acknowledges that this contribution was co-authored by an affiliate of the national government of Canada. As such, the Crown in Right of Canada retains an equal interest in the copyright. Reprints must include clear attribution to ACM and the author’s government agency affiliation. Permission to make digital or hard copies for personal or classroom use is granted. Copies must bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. To copy otherwise, distribute, republish, or post, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

EDBT/ICDT 2017 Joint Conference March 21–24, 2017, Venice, Italy

© 2017 ACM. ISBN XXXXXXXX.

DOI: XXXXXX

However, perfect matching of the labels of all query edges can be too strong constraint and may not retrieve many desired matches. This paper extends exemplar queries with edit distance operations. A detailed motivating example is given in Section 2. Since edges with mismatching labels can match when edit distance operations are allowed, we call our queries error-tolerant exemplar queries (ETEQ).

Since ETEQ generalizes exemplar queries, the queries in ETEQ are applicable in many domains where one does not have a clear idea of what is being searched, but has a starting element in the result set. For example, the query “Ruby and D influenced Swift” with ETEQ can give the influence relationships between programming languages as well as other relevant relationships that may be retrieved when edit operations are allowed. Also, ETEQ can help the existing search engine services improve in two ways. First, search engines can append the results of ETEQ to their results, which can increase their recall and possibility capture users’ information needs. Second, the results of ETEQ can be considered related or additional queries that are suggested by the search engines. For example, when a user searches information about “relationship between Ruby and Swift”, current search engines will show the results that mention the relationship. ETEQ can provide relationships of other programming languages, e.g. Java and Closure, Swift and Ruby, etc. These results can either be considered as related queries that search engines suggest to users, or they can be added to the result set that is returned.

Our contributions can be summarized as follows:

- We extend exemplar queries with edit distance operations to support error-tolerant searches on graph data.
- We propose two efficient algorithms for ETEQ based on a filtering-and-verification framework, and study efficient pruning strategies that use the neighbourhood structure and the paths to filter unqualified results.
- We develop cost models that allow us to compare the cost of our algorithms and across different queries without actually running the algorithms.
- We analyze our algorithms using our cost model and study the conditions under which one algorithm is expected to outperform the other.
- We perform a thorough experimental evaluation of the effectiveness of our filtering schemes, the performance and the scalability of our algorithms and the reliability of our cost model.

2. MOTIVATING EXAMPLE

Consider a search scenario where one wants to find more information about programming languages. The user, if not familiar with the area, may try “programming languages’ basic information”. But this query will most likely return documents discussing programming languages in general terms. The user may instead provide an example result. Thus, he can write a query with all basic information about Swift as shown in Figure 1. This query, typed to a search engine, may return results about Swift or other query keywords (e.g. Ruby, Scala, etc.) without any result covering other languages.

Using exemplar queries allows us find relevant answers matching all query edges. However, the given relationship s in the query only holds for Swift. In other words, there is no other relevant answer that perfectly matches all query edges and their labels. Consider the candidate answer shown in

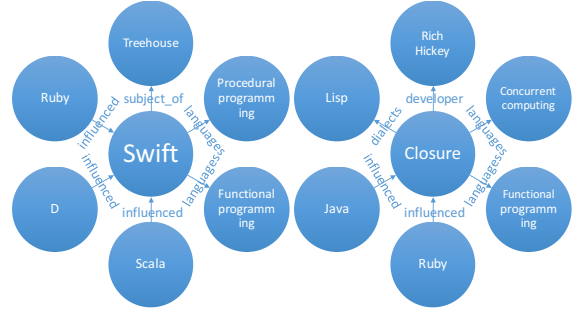


Figure 1: ETEQ and relevant answer

Figure 1. Although the relationships between Closure and Rich Hickey (“developer”) and between Closure and Lisp (“dialects”) do not appear in the query graph, the candidate has very similar structure to the query and is very likely an answer that users will find relevant. Exemplar queries cannot find such relevant answers. Also, it might be difficult for the users to describe the query with accurate relationships between entities, e.g. relationship between Swift and Treehouse.

Thus, there is need to devise a method for searching relevant answers that the user is interested in from a single example possibly with errors and mismatches of labels.

3. DEFINITIONS AND PROBLEM STATEMENT

The “Resource Description Framework” or RDF[10] is a standard data model for describing Web resources, but it also has become a popular data model for publishing linked data. RDF encodes Web data as a labeled directed graph using nodes to represent Web resources or entities and edges to represent the links between them. RDF data can be formally represented as a directed labeled graph.

DEFINITION 1. (RDF Graph) An RDF graph $G = \langle V, E, L \rangle$ is an directed labeled graph, where V denotes a set of nodes, $E \subseteq V^2$ is a set of edges, and L is a labeling function that maps each node in V and each edge in E to a label.

Unless explicitly stated otherwise, the term graph in this paper refers to a labeled directed graph.

DEFINITION 2. (Edge-preserving Isomorphism) A graph G is edge-preserving isomorphic to a graph G' , denoted as $G \simeq G'$, if there is a bijective function μ from the nodes of G to the nodes of G' such that for every edge $n_1 \xrightarrow{l} n_2$ in G , the edge $\mu(n_1) \xrightarrow{l} \mu(n_2)$ in G' .

DEFINITION 3. (Edge-preserving Edit Distance) The edit distance between two non-isomorphic graphs G and G' is the minimum number of of edit operations that makes $G \simeq G'$.

Generally, edit operations include insertion, deletion and substitution of edges or nodes. Our discussions in this paper focus on the substitution of edge labels. However, our algorithms can easily be extended to support other two edit operations.

DEFINITION 4. (Error-tolerant Exemplar Query) An error-tolerant exemplar query is a query that may contain error on the edge labels and may not perfectly match relevant answers.

In the rest of paper, we will use the term query for an error-tolerant exemplar query and the relevant answers to denote the query result set.

DEFINITION 5. (Query Cost Model) A query cost model is a parametric equations that estimates the cost of an algorithm or a query plan, in terms of the number of operations (I/O and CPU) that is needed to evaluate the query.

Problem Statement: We aim to address the following two problems. (1) Given an ETEQ in the form of a query graph q and an edit distance threshold t , we aim to efficiently retrieve all relevant answers in a data graph that are edge-preserving isomorphic to q with at most t edit operations. (2) Given two algorithms for the problem in (1), we aim to compare their costs in terms of a cost model and find out the conditions under which one algorithm outperforms the other.

4. EXEMPLAR QUERIES WITH EDIT DISTANCE CONSTRAINT

4.1 The Basic EXED Algorithm

Given a data graph $G = (V, E)$, a query Q and the edit distance threshold t , the basic exemplar queries with edit distance constraint algorithm (EXED) discovers a set of subgraphs that are within t edit distances of the query q . A naive approach is to compare the query with every subgraph in the data graph G . Instead of comparing the query with an exponential number of subgraphs in the data graph, EXED randomly chooses one node n_q from the query as a seed. Subsequently, it considers all nodes of the data graph one by one as a possible mappings of the node n_q . For each such node n_g in V , it checks if there exists a subgraph that contains n_g and is isomorphic to the query with at most t edit operations. All matching subgraphs are added into the result set. Algorithm 1 describes the above steps in pseudocode.

The algorithm starts from the query subgraph q only containing n_q and a data subgraph g only containing n_g , and maps n_q to n_g . It iteratively adds edges from Q and G to q and g respectively until q is equal to Q and g is isomorphic to Q with at most t edit operations.

Algorithm 1 EXED

Input: Data graph $G = \langle V, E \rangle$, query graph Q

Input: Threshold t

Output: Set of answers S

```

1:  $S \leftarrow \emptyset$ 
2:  $n_q \leftarrow \text{chooseARandomNode}(Q)$ 
3: for each node  $n_g \in V$  do
4:    $s = \text{SEARCHSIMILARSUBGRAPH}(G, Q, n_q, u, t)$ 
5:   if  $s \neq \emptyset$  then
6:     Add  $s$  to answer set  $S$ .
7:   end if
8: end for
9: return  $S$ 
```

4.2 A Neighbourhood-based Pruning

In EXED, every node n_g of the data graph is considered a possible match of the query node n_q and as a seed to

start the search for relevant answers. However, only a small fraction of data nodes are true candidates, and considering every node of the data graph as a possible match of n_q is highly inefficient.

To reduce this search space, one has to reduce the number of unnecessary data nodes from which the search for similar subgraphs starts. Inspired by [9], we propose a method called NEIGHBOURHOODPRUNING to prune the search space.

DEFINITION 6. (d -neighbour) Let $n \in V$ be a node of the data graph $G = \langle V, E \rangle$. The node $n_i \in V$ is a d -neighbour of n if there exists a path from n to n_i of length at most d . The d -neighbourhood nodes of n , denoted as $N_d(n)$, is the set of all d -neighbours of n , and the d -neighbourhood labels of n , denoted as $L_d(n)$, is the set of edge labels on paths of length at most d from n to its d -neighbour nodes.

NEIGHBOURHOODPRUNING compares data nodes with query nodes using their neighbourhood information, and filters out those data nodes that requires more than t edit operations to match the query node's neighbourhood. Let $T_{n,k,l}$ denotes those neighbour nodes of n which are reachable from n in a path of length k and l is the last label in the path, i.e.

$$T_{n,k,l} = \{n_1 | n_1 \xrightarrow{l} n_2 \cup n_2 \xleftarrow{l} n_1, n_2 \in N_{k-1}(n)\}.$$

Since keeping the table of neighbour nodes for every data nodes is expensive in term of space, we only keep the cardinality of $T_{n,k,l}$. Also, in order to efficiently retrieve data node candidates for a query node, we implement an inverted index which stores a list of nodes for every label, every cardinality and every distance. In other words, the index allows us to efficiently find data nodes that have a label l at their k -neighbourhood with a certain cardinality.

Once the neighbourhood tables $T_{n,k,l}$ of both data and query nodes are computed for each label l and path length $k \leq d$, then we can compare the neighbourhood of a query node to that of a data node and filter out unqualified data nodes. The edit distance between data node n_g and query node n_q for label l at k -neighbourhood can be written as

$$\text{dist}_{k,l}(n_q, n_g) = \begin{cases} 0 & \text{if } |T_{n_g,k,l}| \geq |T_{n_q,k,l}| \\ |T_{n_q,k,l}| - |T_{n_g,k,l}| & \text{otherwise.} \end{cases}$$

Given an edit distance threshold t , n_g is considered a candidate for the query node n_q when the distance between the d -neighbourhoods of the two nodes does not exceed t , i.e.

$$\sum_{i=1}^d \sum_{l \in L_i(n_q)} \text{dis}_{i,l} \leq t.$$

Note that this filtering may introduce false positives, because neighbourhood-based pruning cannot identify if the labels are in the same path. For example, this neighbourhood-based distance between query nodes q_1 and g_1 in Figure 2 is 0 whereas the actual edit distance is 6. It may be noted that the more correlated the edge labels in a query's path are, the less false positives the neighbourhood-based pruning can produce. This summarized representation of a neighbourhood is highly effective at pruning nodes without actually visiting their neighbourhood and the false positives can be removed at the verification stage.

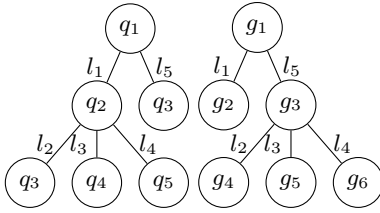


Figure 2: Query graph and data graph

At the verification stage, instead of comparing query nodes with data nodes independently and combining the match results, a more efficient approach is to take the previous comparisons of the nodes into the consideration.

DEFINITION 7. (Simulation) Let $G_1 = \langle V_1, E_1 \rangle$ and $G_2 = \langle V_2, E_2 \rangle$ be two graphs. G_2 simulates G_1 if there exists a relation R such that, for every node $n_1 \in N_1$ and $n_2 \in N_2$ for which $(n_1, n_2) \in R$ and $n_1 \xrightarrow{l} n'_1$, there exists a n'_2 such that $n_2 \xrightarrow{l} n'_2$ and $(n_1, n_2) \in R$.

Verifying a simulation can be done more efficiently since $n'_2 \in V_g$ is a possible match of a query node n'_1 only if in a previous comparison of nodes n_2 and n_1 , n_2 is identified as a possible match of n_1 and there is an edge between n_2 and n'_2 with label l and a corresponding edge with label l between n_1 and n'_1 . With this observation, we only need to examine adjacent nodes of previously matched data nodes rather than all data nodes to find possible matches of a query node.

The EXED algorithm randomly chooses a query node n_q as a seed (starting node) and starts the search from the seed node. However, when we take the previous comparisons of the nodes into consideration, the choice of the starting node can affect the performance of the algorithm. The fewer previously matched data nodes are, the less comparisons we need do in the following steps of the simulation. We implement this by introducing the concept of selectivity into our algorithm.

DEFINITION 8. (Selectivity) The selectivity of a query node n is the ratio of n 's matches to the number of vertexes in G . The selectivity of label l is the ratio of frequency of label l to the number of edges in G .

As the actual selectivity of a query node may be known only after finding its matches, we devise a method to estimate the selectivity in advance. In this section, we assume the selectivities of nodes are known. The selectivity estimation is studied in detail in Section 5.1.

Let n_{min} be a query node with the minimum selectivity. The algorithm initially takes the set of all data nodes as candidate mappings candidate of n_{min} . For each query node n_q that has not been visited yet, the algorithm checks if each data node $n \in V_g$ has the matching edges (i.e. edges with the same label and direction) for each adjacent edges of n_q . If it does not match and the edit distance t has already reached the threshold, (n, t) is removed from $\mu(n_q)$. If it does not match and t has not reached the threshold, a node n' adjacent to n is considered as a candidate for the query node n'_q adjacent to n_q and the entry $(n', t+1)$ is inserted into $\mu(n'_q)$. Otherwise, n' is the candidate of n'_q with no edit penalty and the entry (n', t) is inserted into $\mu(n'_q)$. Finally, the query node n_q is marked as visited and

Algorithm 2 NEIGHBOURHOODPRUNING

Input: Data graph $G = \langle V_g, E_g \rangle$, query graph $Q = \langle V_q, E_q \rangle$

Input: Threshold t

Output: Set of candidate mappings $\mu \subset V_q \times N_g$

```

1:  $L_d^s \leftarrow d$ -neighbour labels of  $Q$ 
2:  $Vis \leftarrow \emptyset$ 
3:  $n_{min} \leftarrow \underset{n \in V_g}{\operatorname{argmin}} Sel(n)$ 
4:  $N_g \leftarrow (V_g, \vec{0})$ 
5:  $\mu(n_{min}) \leftarrow N_g$ 
6:  $Q \leftarrow \{n_{min}\}$ 
7: for each  $n_q \in Q$  do
8:   if  $\langle n_q, n'_q \rangle_l \in E_q$  and  $n'_q \notin Vis$  then
9:     Update edit distance of nodes in  $\mu(n_q), \mu(n'_q)$ .
10:    Remove nodes that exceed threshold.
11:   end if
12:    $Q \leftarrow Q \cup \{n'_q | n_q \xleftarrow{l} n'_q \vee n_q \xrightarrow{l} n'_q\}$ 
13:    $Q \leftarrow Q \setminus \{n_q\}$ 
14:    $Vis \leftarrow Vis \cup \{n_q\}$ 
15: end for
```

is removed. Algorithm 2 describes the pseudocode of above steps.

4.3 Speeding up Neighbourhood-based Solution

The neighbourhood-based filtering may introduce false positives, because two matching labels may not be under the same path or have the same direction. In this section, we introduce a path-based filtering algorithm to prune out some of the false positives.

The path-based filtering algorithm compares data nodes with the query node in terms of their paths and filters out those data nodes that requires more than t edit operations to match the query node. However, keeping every path for every node can be very expensive in term of space. For a graph with average degree \hat{D} and d -edge path indexes, the space required for using path indexes is $O(\hat{D}^d)$. Our approach to reduce the space is using the Bloom filter[2].

A Bloom filter is a space-efficient probabilistic data structure to efficiently test whether an element is a member of a set N . An empty Bloom filter is a bit array of m bits, all set to 0. There are k different hash functions, each mapping an element to one of the m array positions. To query for an element, one needs to find the k array positions the element is mapped to. If any of the bits at these positions is 0, the element is definitely not in the set. If all are 1, then either the element is in the set, or the bits have by chance been set to 1 during the insertion of other elements, resulting in a false positive. The error rate p depends on m , $|N|$ and k . We set the false positive rate to 1%. The optimal number of hash functions is approximately $0.7m/|N|$, and the optimal number of bits m is approximately $|N| \ln p / \ln^2 2$. The number of inserted elements can be estimated by \hat{D}^d , where \hat{D} is the average degree of the data graph[5]. The Bloom filter based path filtering allows us to control the false positives at a low rate with a compact storage and an efficient access time. Moreover, it has no false negatives.

To insert a path into the Bloom filter, we concatenate the labels in the path to form a string that is inserted into the

Bloom filter. To encode the direction of an edge, a sign symbol is added to each label to distinguish between incoming and outgoing edges. In addition, the count of each path is described by preceding the label sequence and separated from the rest of string by “P”. For example, the string “2P+1-2” describes two paths that have one outgoing edge labeled with value 1 and one incoming edge labeled with value 2. Since all labels in a path are encoded into one string, an unmatched path can have up to d unmatched labels. To avoid filtering out false negatives, we consider the lower bound of the edit distance for an unmatched path, which is 1. This also introduces false positives if we only use path filtering. However, these false positives can be removed by considering the neighbourhood filter.

Our experiments show that the two filtering schemes work nicely, complementing each other. This is because path filtering can identify if multiple labels are in the same path and if the matching edges with the same labels have the same direction which neighbourhood filter cannot do; on the other hand, neighbourhood filtering can identify the level of mismatched labels which cannot be done by a path-based filtering.

4.4 A Wildcard Solution

The main problem with EXED is that the number of intermediate results can become huge, especially for large edit distance thresholds and large node degrees of the data graph. Most of those intermediate results need to be kept until a very late stage of the searching.

To reduce the number of intermediate results, we develop a new algorithm referred to as wildcard queries with edit distance constraint (WCED). The approach taken in this algorithm is to map the subgraph edit distance problem instance into subgraph isomorphism problem instances without missing any relevant answers. This is done by introducing the concept of wildcard labels.

DEFINITION 9. (Wildcard Label) A wildcard label is a label that can substitute for any other label in graph matching.

The main idea is to perform multiple subgraph isomorphism searches based on the original query and merge the retrieved answers to obtain the final results. This approach has two phases: query pre-processing and subgraph search and answer mergence.

In the query preprocessing phase, we choose t edges from $|E_q|$ query edges, where t is the edit distance threshold and set their labels to the wildcard label. This gives us $\binom{|E_q|}{t}$ wildcard queries assuming that $t \leq |E_q|$. For example, Figure 3 shows a two-edge query and its wildcard queries with edit distance threshold 1.

In the next phase, we run subgraph isomorphism searches on those generated wildcard queries. For this, we directly adopt EXED with edit threshold set to 0. This returns the subgraphs where the wildcard matches any labels. For example, searching for the first wildcard query in Figure 3 will give us all subgraphs which have an edge labelled l_1 and an edge with any label, both under the same parent node. Finally, duplicates due to possible overlappings between wildcard queries are removed.

The WCED algorithm reduces the number of intermediate results by converting the subgraph edit distance into subgraph isomorphism. This is for the cost of running EXED $\binom{|E_q|}{t}$ times with edit distance threshold 0.

Another advantage of using WCED is that it can be easily extend to handle other edit operations, such as insertion and deletion (as discussed next).

Supporting deletions This is similar to substitution except that the edges are removed instead of being labeled with a wildcard. With $|E_q|$ query edges and edit distance threshold t , there are $\binom{|E_q|}{t}$ possible choices for deletion, each leading to a subgraph isomorphism search.

Supporting insertions Since we are searching for subgraphs of the query graph, insertions are already supported at no cost. For example, the data graph can have any number of additional edges, and those edges are ignored in a subgraph search (and there is no need to insert query edges).

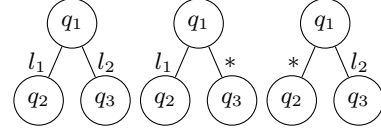


Figure 3: Query graph and its wildcard queries

5. ALGORITHM COST ANALYSIS

In the previous sections, we presented two algorithms for exemplar queries with edit distance constraints, each with some advantages over the other. WCED has much less number of intermediate results (meaning less space usage), while EXED only needs to be run once and has no duplicate answers. To determine which algorithm has the least cost for a given query and data graph (without actually running the algorithms), one needs an accurate cost estimation. This is the problem addressed in this section.

EXED consists of three parts: starting node selection, neighbourhood-based pruning and subgraph verification. The time cost of starting node selection and neighbourhood-based pruning are linear in the number of query nodes and number of data graph nodes respectively, while the time cost of subgraph verification grows exponentially with the edit distance threshold and the number of query edges. WCED consists of three phases: query pre-processing, subgraph isomorphism search and answer mergence. Subgraph isomorphism search uses EXED with the edit distance threshold 0, the cost of which also grows exponentially with the number of query edges. The time cost of query pre-processing depends on the number of query edges and the edit distance threshold. The time cost of answer mergence is linear in the number of answers. Both of them are relatively low and negligible compared to the cost of subgraph isomorphism search. Therefore, we focus on the cost of verification of two algorithms. The cost depends on the number of data nodes (candidates) matching the query starting node and the cost of verifying each candidate.

In this section, we first present an estimation for the selectivities of edge labels. We then present an exact cost model and an upper-bound cost model.

5.1 Selectivity Estimation

The probability that an arbitrary edge in the data graph has label l , referred to as the selectivity of label l , can be written as

$$Sel(l) = \frac{freq(l)}{|E_g|}$$

where $\text{freq}(l)$ is the frequency of label l in the data graph G , and $|E_g|$ is the number of edges in G .

5.2 An Exact Cost Model

Both algorithms EXED and WCED start with a set of candidate nodes in data graph G that are likely to match a query node n_s ; those candidates may be selected based on a filtering scheme such as the neighbourhood or the path filtering. Given a candidate node in G , we must check if there is a subgraph in G that simulates the query graph in which the candidate node matches n_s . The cost of this process depends on two factors: the number of candidates matching the query node and the cost of verifying each candidate.

5.2.1 A cost model for WCED

Given a query and an edit distance threshold that is larger than zero, the WCED algorithm generates a set of wildcard queries based on the edit distance threshold, hence it has to perform multiple subgraph isomorphism searches on those wildcard queries. To estimate the cost, we need to estimate the cost of each search and sum up the costs. It should be noted that a wildcard query is like any query except that some edges are labeled with wildcards and those wildcards can match any label.

Estimating the number of candidates: Given a seed n_s , we want to estimate the probability that a data node is a candidate for n_s .

LEMMA 1. *Given a query node and its adjacent edge labels l_1, \dots, l_k , and assuming independence between the labels, the probability that a data node with D adjacent labels has all query labels is*

$$P_D(l_1, l_2, \dots, l_k) = \sum_{j=1}^{k-1} \sum_{i=j+1}^k (-1)^{i-1} P_D(\neg l_j, \dots, \neg l_{i-1}, l_{i+1}, \dots, l_k) + \sum_{j=1}^{k-1} (-1)^{k-j+1} (1 - \sum_{i=j}^k \text{Sel}(l_i))^D + (1 - (1 - \text{Sel}(l_k))^D). \quad (1)$$

See the extended version of this paper [14] for a detailed proof. Lemma 1 directly gives the selectivity of a query node based on its 1-neighbourhood. Let $L_i(n_q)$ denote the set of labels at the i^{th} neighbourhood of a query node n_q . The probability that the neighbourhood of a data node matches that of a query node at levels $1, \dots, d$ can be written as

$$P(n_q) = \prod_{m=1}^d P_{D_m}(L_m(n_q)). \quad (2)$$

where $P_{D_m}(L_m(n_q))$ is as defined in Lemma 1 and D_m is the number of edges at the m^{th} neighbourhood of a data node. We generally don't know D_m when estimating our probabilities in Equations 1. Assuming that each data node has the same degree \hat{D} , then $D_m = \hat{D}^m$. Then, we can have the number of candidates matching query node n_q as

$$|C(n_q)| = |V_g| * P(n_q).$$

Estimating the cost of verifying each candidate: For each candidate of the starting node, the algorithm starts from a graph g with only one node (i.e. the candidate node) and iteratively adds new edges to g until either g simulates the query, or no such simulation is found. The cost of adding

each new edge depends on the expected number of matching edges of a query edge and the number of subgraphs to which the edges are added. Let \hat{D} denote the expected degree of a data node. For a query label l_i , we expect $\hat{D} * \text{Sel}(l_i)$ edges of a node in the data graph to match l_i . For a fixed candidate node in the data graph, the expected number of subgraphs (partial matchings) that can be constructed starting from the candidate and simulating the query subgraph rooted at the seed with labels l_1, \dots, l_k is

$$\prod_{i=1}^k \hat{D} * \text{Sel}(l_i).$$

and the total expected cost of verifying a candidate n is

$$\sum_{i=1}^{|E_q|} \prod_{j=1}^i \hat{D} * \text{Sel}(l_j). \quad (3)$$

Note that this is based on the assumption that a search starting from a candidate node will not stop early if the simulation exceeds the edit distance threshold.

The total cost of verifying $|C(n_q)|$ candidates for possible matches to query q is

$$\text{Cost}(q) = |C(n_q)| * \sum_{i=1}^{|E_q|} \prod_{j=1}^i \hat{D} * \text{Sel}(l_j). \quad (4)$$

Since we have replaced a query graph with $\binom{|E_q|}{t}$ graphs each with t wildcards, the total cost is the sum of the costs of verifying those wildcard queries.

5.2.2 EXED Cost Model

To estimate the cost for EXED, we also need to estimate the number of candidates in the data graph matching a query seed node and the cost of verifying each candidate.

Estimating the number of candidates: Since a data node is allowed to have up to t edit operations in its neighbourhood, directly estimating the probability that a data node is a qualified candidate is difficult. Therefore, we estimate the number of candidates for a set of wildcard queries where the labels are all fixed. By summing up the number of candidates for these wildcard queries and removing the repetitive candidates due to overlaps between queries, the number of candidates for n_q in EXED can be written as

$$|C(n_q)| = \sum_{i=1}^{\binom{|E_q|}{t}} |V_g| * P(n_{w_i(q,t)}) - \left(\binom{|E_q|}{t} - 1 \right) * |V_g| * P(n_q). \quad (5)$$

where $w_i(q, t)$ is a wildcard query constructed from q by replacing t edge labels with wildcards and $P(n_q)$ is as in Equation 2. With t edit operations, the number of wildcard queries is $\binom{|E_q|}{t}$ and the last term gives the number of double-counts.

Estimating the cost of verifying each candidate: To estimate the cost of verifying each candidate, we need to estimate the number of partial matchings. There are two kinds of partial matchings in EXED: (1) those matchings that have reached the edit distance threshold; (2) those matchings that have not reached the threshold. For those partial matchings that have not reached the threshold, edges with any label

can be added to the matching in the next step of the simulation, whereas for those matching that have reached the threshold, only edges with matching labels can be added. Let m be the number of edges in a partial matching, and k be the number edges in the matching where the matching edges have different labels. If l_1, \dots, l_k denote the query labels in the matching where the labels don't match, and l_{k+1}, \dots, l_m be the labels where both data and query edges in the matching have the same labels, then the number of partial matchings can be written as

$$\hat{D}^m \prod_{i=1}^{m-k} Sel(l_i) \prod_{j=1}^k (1 - Sel(l_j)).$$

Given query labels l_1, \dots, l_m , we generally don't know in advance which labels will mismatch and need to check all choices of $\binom{m}{t}$ sets of labels. The number of partial matchings that needs to be verified is

$$S_t(q, m) = \begin{cases} 0 & \text{if } t > m \\ \hat{D}^m \prod_{i=1}^m Sel(l_i) & \text{if } t = 0 \\ \sum_{k=1}^{\binom{m}{t}} \hat{D}^m \prod_{i=1}^{m-t} Sel(l_{k,i}) \prod_{j=1}^t (1 - Sel(l_{k,j})) & \text{if } t < m. \end{cases} \quad (6)$$

For those partial matchings that have not reached the threshold t , any edge can be added into those partial matchings in the next step of the simulation. In this case, the next step of simulation costs

$$\sum_{j=0}^{t-1} S_j(q, m) * \hat{D}.$$

For those partial matchings that have reached the threshold, only edges with a matching label can be added into those partial matchings. In this case, the next step of simulation costs

$$S_t(q, m) * \hat{D} * Sel(l_{m+1}).$$

The cost of verifying each candidate in EXED can be written as

$$Cost(q) = \sum_{i=0}^{|E_q|-1} (S_t(q, i) * \hat{D} * Sel(l_{i+1}) + \sum_{j=0}^{t-1} S_j(q, i) * \hat{D}). \quad (7)$$

The total cost of EXED is the product of the number of candidates (as given in Equation 5) and the cost of verifying a candidate (as given above).

$$Cost_{ex} = |C(n_q)| Cost(q).$$

5.2.3 Cost Model Comparison

Our cost comparison assumes that the edit threshold t is less than the number of query edges; otherwise, the problem is subgraph isomorphism with no label constraints, which is not addressed in this paper. With that assumption, we want to compare the costs of verifying the candidates for EXED and WCED and find out the conditions under which one outperforms the other.

When the edit distance threshold is larger than zero, the cost of verifying a candidate in EXED is higher than that in WCED, because edit operations can happen on any label in EXED while they are fixed in WCED. In this case,

when the number of candidates for WCED and EXED are roughly the same, WCED will outperform EXED. In other words, WCED outperforms EXED if the number of candidates for the original query is small (See Equation 5). With few candidates we expect to have fewer isomorphic matches, and this a scenario studied in this paper where matches with edit distance larger than zero are relevant. The next lemma shows what happens when this condition does not hold.

LEMMA 2. *Given a data graph with expected node degree \hat{D} , a query graph q with at least 2 edges and the edit distance threshold set to 1, the cost of verifying a candidate in EXED is less than the sum of the cost of verifying a candidate for every wildcard queries in WCED when*

$$Sel(l_1) > \frac{1}{|E_q| \sqrt{\hat{D}}}. \quad (8)$$

where l_1 is a query label that has the highest selectivity (i.e. the smallest value of $Sel(l_i)$).

See the extended version of this paper [14] for a detailed proof. When the number of candidates for the original query is very large and is approximately equal to the number of candidates for each wildcard query, the cost of EXED and WCED can both be approximated by the number of candidates for the original query and the cost of verifying each candidate. In this case, EXED can outperform WCED given the condition of the lemma.

5.3 An Upper Bound Cost Model

The exact cost model is based on two assumptions: (1) labels are evenly distributed, and (2) labels are pairwise independent. These assumptions may not stand in real-world data graphs. The error due to these two assumptions will accumulate and become significant as the number of query edges increases. In this case, we need another cost estimation to deal with larger queries.

In this section, we present a cost model that gives an upper bound of the actual cost but is more accurate for larger query graphs.

Estimating the number of candidates: To estimate the upper bound for the number of candidates, two weaker assumptions of label independence are considered: (1) the labels of the adjacent edges of a data node are independent whereas labels, which are in a path starting from a node, are correlated; (2) the labels of the adjacent edges of a data node are correlated whereas labels, which are in a path starting from the node, are independent. For two or more correlated labels, the selectivity of the label with the least selectivity provides an upper bound of the selectivity of the set.

Under the first assumption, the selectivity of the label with the minimum selectivity in each path is used to estimate the selectivity upper bound of the path. This reduces each path in the query to an edge (with the minimum selectivity), and as a result the query becomes a node with a set of adjacent edges (i.e. a tree with only one level). Assuming independence between the labels of these edges, Lemma 1 will give an upper bound of the probability that a data node is a candidate for a query node. Note that D in the Lemma is set to the number of paths in the d -neighbourhood.

Under the second assumption, all edges under a node are collapsed into a single edge, which is labeled with a label from the set that has the least selectivity. Since the edge

labels of the resulting query are all independent, Equation 2 can be used to estimate the upper bound.

Estimating the cost of verifying each candidate: To estimate the upper bound for the cost of verifying each candidate, the maximum frequency of each label under a node is used to upper bound the number of matching label in each step of the simulation. Let $N(l_i)$ denote the maximum frequency of label l_i in the adjacent edges of a node.

In our exact cost model, the number of matching labels for a label l_i is $\hat{D} * Sel(l_i)$ assuming that every label is uniformly distributed on the adjacent edges of a node. Replacing $\hat{D} * Sel(l_i)$ in Equations 4 and 7 by $N(l_i)$ will give us an upper bound of the cost of verifying each candidate in WCED and EXED respectively.

6. EXPERIMENTAL EVALUATION

This section presents an experimental evaluation of our algorithms and cost models. All experiments were performed on a 2.4 GHz 4 Core CPU with 60G memory running Linux. The algorithms are implemented in Java 1.8. In our experiments, unless explicitly stated otherwise, the path length d in our filtering scheme is set to 3.

Dataset: We downloaded a full dump of Freebase¹ in May 2015. We removed the triples that were used as internal specification for the community (e.g. user and group data and discussion topics) obtaining a fully connected graph of 84 million nodes and 335 million edges. Since the entire Freebase is too large for our machine (occupies approximately 90G of memory when fully loaded), we extract sub-graphs from Freebase with different parameters. The sub-graphs are extracted using a breadth first traversal of the graph from a randomly selected starting node and randomly choosing new edges to be included in the data graph. Unless explicitly stated otherwise, the data graphs are randomly generated from Freebase with the number of nodes set to 10K and average node degree set to 15.

Queries: Two types of queries are used in our experiments: (1) A set of real queries from the AOL query log, manually mapped to the data graph, and (2) randomly selected sub-graphs of the data graph. These queries vary in the numbers of edges and the selectivities of their labels. Unless explicitly stated otherwise, our experiments use 100 randomly selected queries, each a subgraph of the data graph.

Summary of our experiments: Section 6.2 evaluates the effectiveness of our filtering schemes under different parameter settings. The impact of our filtering schemes on the performance of our algorithms and improvements over existing algorithms are evaluated in Sec 6.4. In Section 6.1, we evaluate our cost models by studying the correlation between our estimated costs and the real costs.

6.1 Effectiveness of Our Cost Models

In this section, we evaluate our cost models in terms of the correlation between our estimates and the actual costs. Our results show that: (1) the selectivity estimation is reliable when the number of query edges $|E_q| \leq 10$; (2) there is a linear relationship between our exact cost model and the real cost for $|E_q| \leq 3$, which allows us to estimate the running time of our algorithms; (3) the exact cost is reliable for the comparison of our algorithms when $|E_q| \leq 6$; (4) the

exact cost is reliable for the comparison of query costs when $|E_q| \leq 8$.

6.1.1 Effectiveness of the selectivity estimation

Since selectivity estimation is a core component of our cost model, we first assess the quality of our selectivity estimation. To do so, we measure the correlation between the actual number of candidates and the estimated number of candidates using the selectivity. In our case, the selectivity is used in choosing a query starting node and for cost comparisons, hence, a relative ordering of the selectivity values is sufficient in these cases. Therefore, we chose Spearman's rank correlation between estimate and actual selectivities, which shows the monotonic relationship of the two variables. The experiments are in the context of WCED and EXED algorithms. Let "exact" denote the exact selectivity estimation, "ub-path" and "ub-adj" denote the upper bound of the selectivity estimations respectively assuming that path labels and adjacent labels are independent. Figure 4 shows that although "exact" has the better correlation (0.96) than "ub-adj" and "ub-path" for queries with 2 edges, the correlation decreases rapidly for queries with a large number of edges, with about 0.55 correlation for queries with 10 edges. In contrast, the upper-bound selectivity estimations remain stable with different numbers of query edges in both WCED and EXED (with correlation between 0.7 and 0.96 and significance level between $3.08E-64$ and $5.38E-16$). We conclude that there is strong positive correlation between the selectivity estimation and the actual selectivity and that we can safely use this selectivity estimation for the choice of a query starting node and cost comparisons.

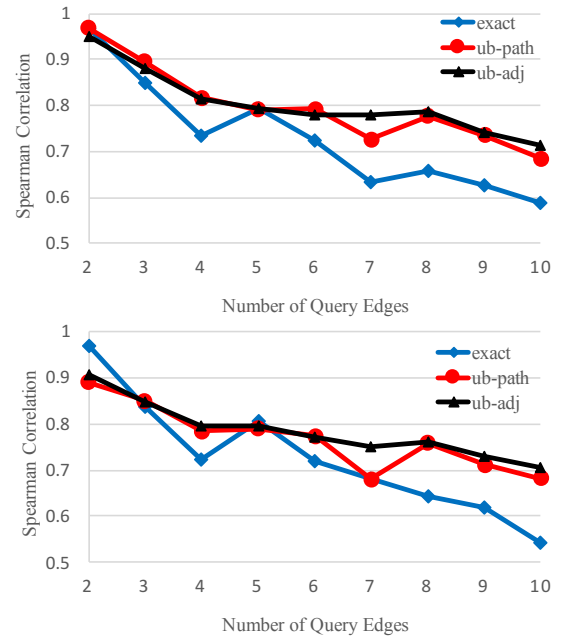


Figure 4: Correlation between estimated and actual selectivities for WCED (top) and EXED (bottom)

6.1.2 Exact cost model evaluation

In this set of experiments, we examine the linear relationship between our estimated cost and the actual number of operations using Pearson correlation. The larger absolute value of the coefficient, the stronger the relationship between

¹<https://developers.google.com/freebase/>

the actual cost and the estimated cost. If the absolute value of coefficient is large, with simple linear regression, we can predict the actual cost from our estimated cost. Figure 5 shows that for small queries (with up to 3 edges), the correlation coefficient is over 0.7 and 0.6 for WCED and EXED respectively. However, the correlation coefficient drop sharply as the number of edges increases. These results are expected because the exact cost model is based on the assumption that the labels are evenly distributed and that they are independent.

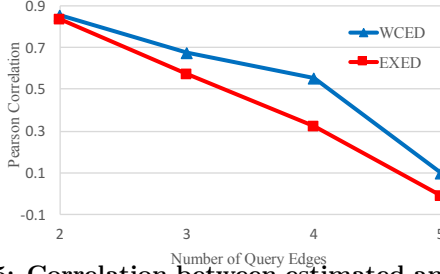


Figure 5: Correlation between estimated and actual costs varying the number of query edges

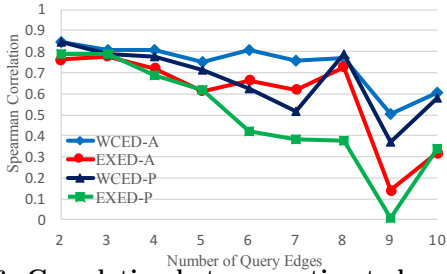


Figure 6: Correlation between estimated and actual costs varying the number of query edges

6.1.3 Upper-bound cost model evaluation

In this set of experiments, we evaluate the upper bound of cost models of our algorithms presented in Section 5.3. Let’s denote with “WCED-A” and “EXED-A” the upper bounds of the cost model of WCED and EXED assuming independence of adjacent labels respectively, and denote with “WCED-P” and “EXED-P” the upper bounds of the cost model of WCED and EXED assuming independence of path labels respectively. Figure 6 shows that both “WCED-A” and “EXED-A” have a better Spearman correlation with the actual cost than both “WCED-P” and “EXED-P”. Both “WCED-A” and “EXED-A” have over 0.6 correlation for queries up to 8 edges, while “WCED-P” only has 0.5 correlation when queries have 7 edges and the correlation of “EXED-P” drops below 0.4 when queries have more than 6 edges. Based on these results, both WCED and EXED using our “WCED-A” provide good cost models for comparing the cost of different queries.

Sometimes we have a query and want to find the algorithm that has the least cost before running the algorithms. To evaluate the effectiveness of our cost models, we computed the gaps between the actual costs of WCED and EXED, i.e. $actW - actE$ and their estimated costs, i.e. $estW - estE$. A high correlation between the two gaps indicates that the cost model can show which algorithm has the least cost even though the actual value of the estimate may not be accurate.

Figure 7 shows us that for queries with up to 6 edges, the correlation coefficient is good (over 0.7).

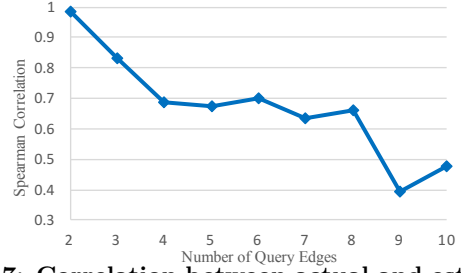


Figure 7: Correlation between actual and estimated cost differences varying the number of query edges

6.2 Effectiveness of Our Filtering Strategies

To evaluate the pruning power of our filtering schemes, the number of nodes in the data graph was set to 10K and the edit distance threshold was set to 1. Let “neighbour” denote the neighbourhood-based filtering strategy, “path” denote the path-based pruning strategy and “both” denote the case where both schemes were used.

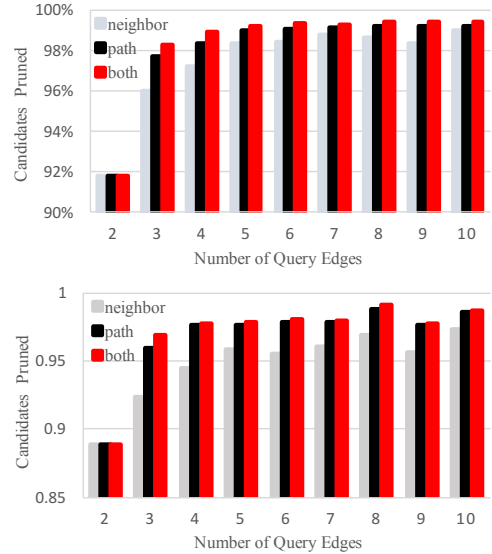


Figure 8: Pruning power of WCED (top) and EXED (bottom) varying the number of query edges

Varying the number of query edges: For this experiment, we varied the number of query edges from 2 to 10 and set the edit distance threshold to 1. Figure 8 shows the fraction of candidates that are pruned in EXED and WCED as we vary the number of query edges. As shown for WCED and EXED, “path” can filter out respectively up to 99.4% and 99.1% of the data nodes on average, while “neighbour” can filter out respectively up to 99.0% and 97.3% of the data nodes on average. The pruning power does not increase by more than 1% when both strategies are used. However, considering the large number of data nodes and the high cost of verifying each candidate, even a small improvement in the pruning stages is amplified and positively affects the performance of the algorithms (See Figure 13).

Varying the edit distance threshold: For this experiment, the number of query edges was fixed at 8 with the edit

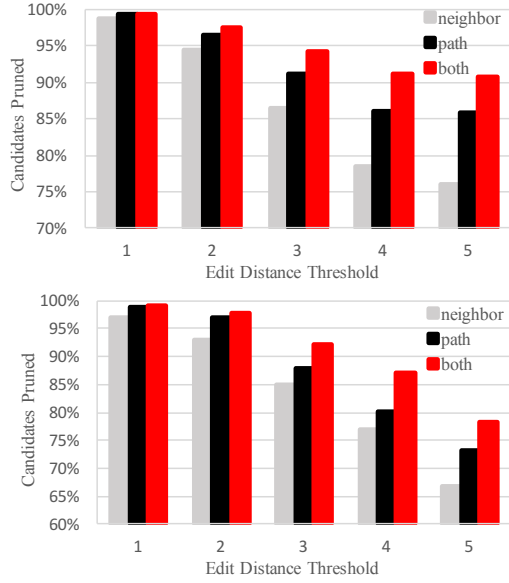


Figure 9: Pruning Power of WCED (top) and EXED (bottom) for different edit distance thresholds

distance threshold varied from 1 to 5. When the edit distance threshold is equal to or exceeds the number of query edges, the labels become irrelevant and the problems becomes subgraph isomorphism on unlabeled graphs, which is not the problem addressed in this paper. Figure 9 shows that both “neighbour” and “path” have good pruning power (over 78%) under different distance thresholds, and it becomes more effective to apply both filtering schemes as the edit distance thresholds increases. This is because “neighbour” scheme does not encode edge direction in its indexes and higher edit distance threshold introduces more false positives with wrong edge direction, while adding “path” on top of “neighbour” can effectively prune out those false positives.

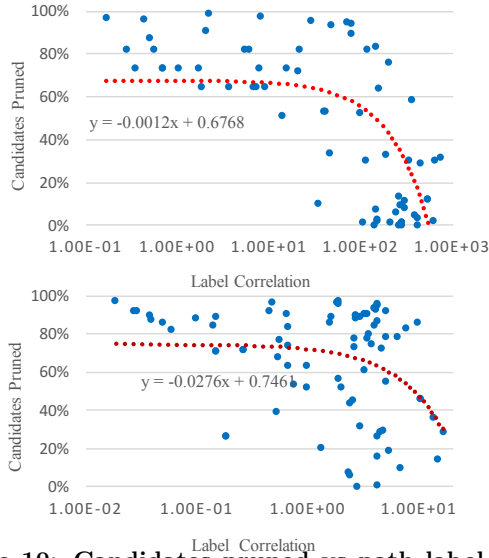


Figure 10: Candidates pruned vs path label correlation of WCED (on the top) and EXED (on the bottom)

Varying path label correlation: For this experiment, the

neighbourhood filtering scheme is considered as a baseline, on top of which we added our path filtering scheme and monitored the improvement in pruning power. We fixed the number of query edges at 8 and set the edit distance threshold to 1. As shown in Figure 10, the improvement in pruning power by adding “path” in both EXED and WCED drops with more correlation. This meets our expectation since the more correlated the labels are, the less false positives the neighbourhood-based pruning can produce and the less room for “path” filtering improvements.

6.3 Combining Filtering Schemes

In these set of experiments, we evaluate the impact of adding path filtering on top of the neighbourhood filtering. In order to show the impact of using both filtering schemes, we consider EXED with the neighbourhood filtering scheme as our baseline and compare it against EXED with both filtering schemes, WCED with the neighbourhood filtering scheme and WCED with both filtering schemes. We denote EXED and WCED with the neighbourhood filtering scheme as “neighbour-EXED” and “neighbour-WCED” respectively, WCED and EXED with both filtering schemes as “both-WCED” and “both-EXED” respectively.

Varying the edit distance threshold: We varied the edit distance threshold t from 1 to 5 and fixed the number of query edges at 8. Figure 11 shows that “neighbour-WCED” speeds up EXED by a factor of 1.5 when $t = 5$, reducing the search time more than half in this particular experiment. Comparing “neighbour-WCED” and “both-WCED”, we find that even though there is no clear speedup for adding path filtering on top of the neighbourhood filtering scheme at small thresholds ($t \leq 2$), the performance gap becomes wider at larger thresholds with around 200 seconds saved when $t = 5$. This meets our expectation because the benefits of using both schemes over “neighbour” in pruning power becomes clear when edit distance threshold increases (See Figure 9).

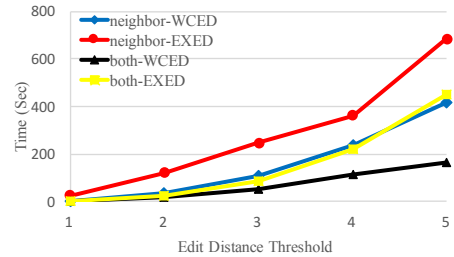


Figure 11: Time vs edit distance threshold

Varying average degree of data graph: In another experiment, we varied the average degree of a node from 5 to 25. Figure 12 shows that “both-WCED” has the greater advantage in a data graph with larger average degrees, outperforming “neighbour-WCED” and “neighbour-EXED”. This is because the cost of verifying each candidate depends on the average degree of the data graph, and a larger average degree results in a higher cost of verifying each candidate and thus the wider gap between “both-WCED” and the others.

Varying the number of query edges: In another experiment, we varied the number of query edges from 2 to 10 with the edit distance threshold fixed at 1. Figure 13 shows that the gap between “neighbour-WCED” and “both-WCED” (and similarly between “neighbor-EXED” and “both-EXED”) widens as we increase the number of edges. This

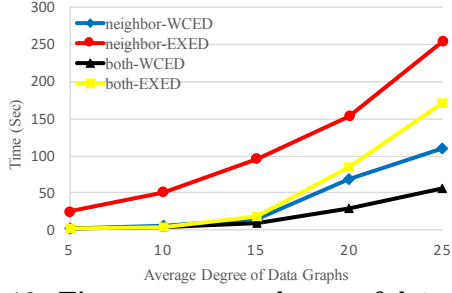


Figure 12: Time vs average degree of data graphs

meets our expectation, because the cost of verifying each candidate grows exponentially with the number of edges.

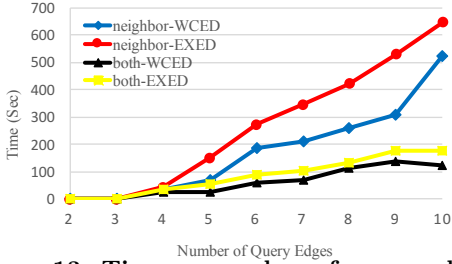


Figure 13: Time vs number of query edges

Our experiments in this section reveals that adding path filtering improves the performance of both algorithms EXED and WCED under one or more of these conditions: (1) the data graph has high average degree; (2) the edit distance threshold $t \geq 2$; (3) the query has over 5 edges.

6.4 Algorithms Comparison

To evaluate the performance of our algorithms against the competitors, we selected two recent algorithms from the literature: (1) SAPPER [19] which is an algorithm for indexing and approximate matching in large graphs, and (2) Exemplar [12] which is similar to our work but is limited to edit distance threshold zero.

Comparing against SAPPER: In this experiment, we compare the scalability of our algorithms against SAPPER. We varied the number of nodes in the data graph from 1K to 100K and set the edit distance threshold to 1. This is consistent with the settings by the authors of SAPPER except that their largest data graph had only 10K nodes. Since SAPPER only supports edge deletion (missing edges), we modified SAPPER to support edge label substitutions. The running time for SAPPER and our algorithms is reported in Figure 14. The Figure shows that the running time of SAPPER grows much faster than our algorithms. We also observe that our algorithms are not very sensitive to the changes in the size of the data graph. WCED with both filtering schemes shows the best performance.

Comparing against Exemplar queries: To evaluate the effectiveness of our queries and to compare them to the exemplar queries of Motin et al. [12], we chose 10 queries from the AOL query log and manually mapped them to subgraphs in freebase. The chosen queries are listed in the Appendix of [14]. The number of query edges ranged between 4 and 8. To control the size of the answer set (and to avoid a blow-up), we varied the edit distance threshold from 0 to 2. When the edit distance threshold is 0, our queries are identical to

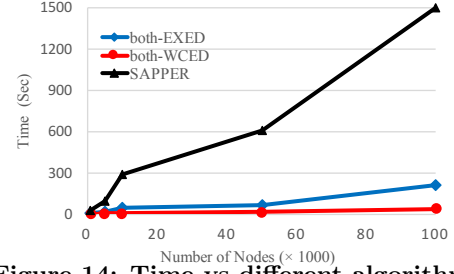


Figure 14: Time vs different algorithms

exemplar queries. As expected and shown in Figure 15, the larger the edit distance thresholds are, the more answers are returned.

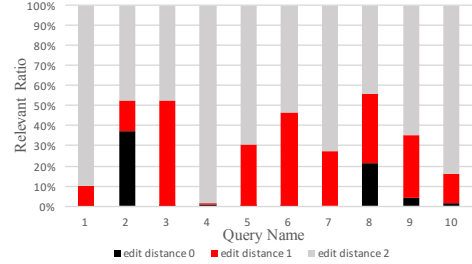


Figure 15: Answer set composition

To evaluate the quality of ETEQ answers, we conducted the following user study. We asked 10 users (uniformly distributed with respect to education level, age and country) to evaluate our system. For each query in the test set, we provided an explanation of the topic, the query intention, and our answer set with different edit distances. We asked each user to rate each result as irrelevant or relevant with respect to the topic and the expressed query intent. Due to the large size of the answer sets, for each answer set and each edit distance, we randomly chose up to 10 answers for evaluation. Figure 16 shows that the ratio of relevant answers to all returned answers decreases as the edit distance threshold increases. This meets our expectation, since the more edit operations are allowed, the less number of relevant answers are expected to be returned. We also observe in Figure 17 that the relevant set has many answers with edit distances 1 and 2. These answers cannot be returned by exemplar queries.

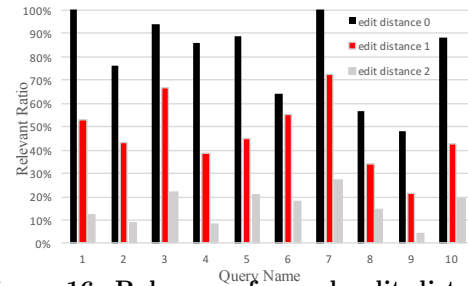


Figure 16: Relevance for each edit distance

We are not comparing the efficiency of our algorithms against Exemplar because (1) Exemplar does not support edit distance thresholds larger than zero, and (2) our EXED algorithm becomes identical to Exemplar when the edit distance threshold is zero and we have extensively evaluated EXED with different edit distance thresholds.

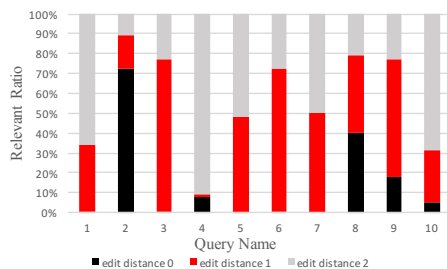


Figure 17: Relevant answer set composition

7. RELATED WORK

The problem of subgraph isomorphism is NP-complete and this is a special case of graph edit distance with the edit distance threshold set at zero. Most existing methods adopt a filter-and-verification framework. Wang et. al.[17] propose an efficient index for sparse data graphs. They decompose graphs to small grams (organized by κ -Adjacent Tree patterns) and uses these κ -AT patterns to estimate a lower bound of their edit distance for candidate filtering. Zeng et. al. [18] propose another method to compute the edit distance by transforming a graph to a multi-set of star structures. Zheng et. al.[18] proposed a path-based index for candidates filtering. However, all these algorithms are targeting data graphs that are small ($< 10K$ nodes), sparse and are not suitable for discovering information from RDF data graphs such as freebase.

Approximate graph matching or similarity based search in large graphs has been studied in the past under various settings. TALE[16] introduces a neighbourhood based index (NH-Index) where it matches important vertices of a query graph first before extending the match progressively. SAPPER[19] takes advantage of pre-generated random spanning trees and a carefully designed graph enumeration order to find approximate subgraph matches. Mongiovi et. al.[11] introduced a set-cover based inexact subgraph matching technique, called SIGMA. Most of these algorithms are designed to find part (e.g. Top-k) of approximate matches and may use their own similarity measures.

Our approach is related to works in which the query node’s neighbourhood information is used to prune graph candidate nodes that are not related to those in the query[9]. Our experiments confirm that combining both indexes performs better than either index alone. Our work is also related to exemplar queries in the way that we both find relevant answers with similarity measure based on edge label. Exemplar queries can only find relevant answers that are edge-isomorphic to the query, while our algorithms in this paper can find relevant subgraphs that are edge-preserving isomorphic to the query after some edit operations.

8. CONCLUSION

In this paper, we study the problem of error-tolerant exemplar queries on RDF graph. Unlike exemplar queries that supports only exact matching of the labels, our developed algorithms allow errors in query and data graph. Two filtering techniques (neighbourhood and path filtering) and two algorithms (EXED and WCED) are developed to handle edit operations as well as to facilitate the searching process. Through a comprehensive experimental evaluation on real and synthetic datasets, we show that our algorithms

are both efficient and effective, outperforming existing algorithms. As a future work, we plan to efficiently support Top-k queries.

References

- [1] A. T. Balaban. Applications of graph theory in chemistry. *Journal of Chemical Information and Computer Sciences*, 25(3):334–343, 1985.
- [2] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
- [3] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proc. of the SIGMOD Conf.*, pages 1247–1250, 2008.
- [4] H. Bunke and K. Shearer. A graph distance metric based on the maximal common subgraph. *Pattern recognition letters*, 19(3):255–259, 1998.
- [5] B. Chazelle, J. Kilian, R. Rubinfeld, and A. Tal. The bloomier filter: an efficient data structure for static support lookup tables. In *Proc. of the SODA Conf.*, pages 30–39, 2004.
- [6] B. Dost, T. Shlomi, N. Gupta, E. Ruppín, V. Bafna, and R. Sharan. Qnet: a tool for querying protein interaction networks. *Journal of Computational Biology*, 15(7):913–925, 2008.
- [7] M.-L. Fernández and G. Valiente. A graph distance metric combining maximum common subgraph and minimum common supergraph. *Pattern Recognition Letters*, 22(6):753–758, 2001.
- [8] X. Gao, B. Xiao, D. Tao, and X. Li. A survey of graph edit distance. *Pattern Analysis and applications*, 13(1):113–129, 2010.
- [9] A. Khan, N. Li, X. Yan, Z. Guan, S. Chakraborty, and S. Tao. Neighborhood based fast graph search in large networks. In *Proc. SIGMOD Conf.*, pages 901–912, 2011.
- [10] G. Klyne and J. J. Carroll. Resource description framework (rdf): Concepts and abstract syntax. W3C, 2006.
- [11] M. Mongiovi, R. Di Natale, R. Giugno, A. Pulvirenti, A. Ferro, and R. Sharan. Sigma: a set-cover-based inexact graph matching algorithm. *Journal of bioinformatics and computational biology*, 8(02):199–218, 2010.
- [12] D. Mottin, M. Lissandrini, Y. Velegrakis, and T. Palpanas. Exemplar queries: Give me an example of what you need. *PVLDB*, 7(5):365–376, 2014.
- [13] J. W. Raymond, E. J. Gardiner, and P. Willett. Rascal: Calculation of graph similarity using maximum common edge subgraphs. *The Computer Journal*, 45(6):631–644, 2002.
- [14] Z. Shao, D. Rafiei, and T. Palpanas. Error-tolerant exemplar queries on rdf graphs. arxiv:1609.03095 [cs-DB], 2016.
- [15] E. Spertus, M. Sahami, and O. Buyukkokten. Evaluating similarity measures: a large-scale study in the orkut social network. In *Proc. of the KDD Conf.*, pages 678–684, 2005.
- [16] Y. Tian and J. M. Patel. Tale: A tool for approximate large graph matching. In *Proc. of the ICDE Conf.*, pages 963–972, 2008.
- [17] G. Wang, B. Wang, X. Yang, and G. Yu. Efficiently indexing large sparse graphs for similarity search. *TKDE*, 24(3):440–451, 2012.
- [18] Z. Zeng, A. K. Tung, J. Wang, J. Feng, and L. Zhou. Comparing stars: on approximating graph edit distance. *PVLDB*, 2(1):25–36, 2009.
- [19] S. Zhang, J. Yang, and W. Jin. Sapper: Subgraph indexing and approximate matching in large graphs. *PVLDB*, 3(1-2):1185–1194, 2010.

APPENDIX

A. PROOF OF LEMMA

A.1 Proof of Lemma 1

PROOF. This lemma can be proved using the probability subtraction rule:

$$\begin{aligned}
P_D(l_1, l_2, \dots, l_k) &= \\
P_D(l_2, \dots, l_k) - P_D(\neg l_1, l_2, \dots, l_k) &= \\
P_D(l_2, \dots, l_k) - (P_D(\neg l_1, l_3, \dots, l_k) - P_D(\neg l_1, \neg l_2, \dots, l_k)) &= \\
P_D(l_2, l_3, \dots, l_k) - P_D(\neg l_1, l_3, \dots, l_k) + P_D(\neg l_1, \neg l_2, l_4, \dots, l_k) &= \\
- P_D(\neg l_1, \neg l_2, \neg l_3, \dots, l_k) &= \\
\vdots &= \\
\sum_{i=2}^k (-1)^{i-1} P_D(\neg l_j, \dots, \neg l_{i-1}, l_{i+1}, \dots, l_k) &= \\
+ (-1)^k P_D(\neg l_1, \neg l_2, \dots, \neg l_k) + P_D(l_2, l_3, \dots, l_k). & \quad (9)
\end{aligned}$$

If we expand $P_D(l_2, \dots, l_k)$ further using the equation above, we will have a set of terms that look similar to the first and the second terms in Eq 9 and the base case $P_D(l_k)$. For the base case, we have $P_D(l_k) = (1 - (1 - \text{Sel}(l_k)))^D$. We also know that $P_D(\neg l_j, \dots, \neg l_k) = (1 - \sum_{i=j}^k \text{Sel}(l_i))^D$ assuming independence. Putting these pieces together will give the statement of the lemma.

□

A.2 Proof of Lemma 2

PROOF. Using Equations 3, the cost of verifying each wildcard queries for WCED with edit distance 1 can be written as

$$Cost_{wc} = \sum_{k=1}^{|E_q|} \sum_{i=1}^{|E_q|} \prod_{j=1}^i \hat{D} * \text{Sel}(l_{k,j})$$

Let l_1, \dots, l_k denotes the labels in increasing order of selectivities. Since the edges in a query are verified in increasing order of selectivities, for those wildcard queries where $l_j (1 \leq j \leq i)$ is not set to wildcard, the edge with label l_i is verified at i^{th} step of the simulation, the cost of verifying the edge is $\hat{D}^i \prod_{j=1}^i \text{Sel}(l_j)$, there are $|E_q| - i$ these type of wildcard queries; for those that $l_j (1 \leq j \leq i)$ is set to wildcard, the edge with label l_{i+1} is verified at i^{th} step of the simulation, the cost of verifying this edge is $\hat{D}^i \prod_{j=1}^{i+1} \text{Sel}(l_j)$, where $l_j \neq l_m$.

Let T_i be

$$T_i = \sum_{k=1}^i \prod_{m=1}^i \text{Sel}(l_{k,m}) \text{ where } l_{k,m} \neq l_k. \quad (10)$$

The sum of verifying costs for those wildcard queries that $l_m (1 \leq m \leq i-1)$ is set to wildcard at i^{th} step of the simulation is $\hat{D}^i (T_{i+1} - \hat{D}^i \prod_{j=1}^{i-1} \text{Sel}(l_j))$. Then, the sum of verifying costs for WCED can be written as

$$\begin{aligned}
Cost_{wc} &= \sum_{i=1}^{|E_q|-1} \hat{D}^i ((|E_q| - i - 1) \prod_{j=1}^i \text{Sel}(l_j) + T_{i+1}) \\
&+ \hat{D}^{|E_q|} T_{|E_q|}.
\end{aligned}$$

Using Equations 6 and 7, the verifying cost of EXED with edit distance threshold 1 can be written as

$$\begin{aligned}
Cost_{ex} &= \hat{D} + \sum_{i=2}^{|E_q|} (\hat{D}^i \sum_{k=1}^{i-1} (1 - \text{Sel}(l_k)) \prod_{j=1}^i \text{Sel}(l_{k,j}) \\
&+ \hat{D}^i \prod_{j=1}^{i-1} \text{Sel}(l_j)), \text{ where } l_{k,j} \neq l_k
\end{aligned}$$

Replacing the terms in above equation with Equation 10, $Cost_{ex}$ can be written as

$$Cost_{ex} = \hat{D} + \sum_{i=2}^{|E_q|} \hat{D}^i (T_i - (i-1) \prod_{j=1}^i \text{Sel}(l_j))$$

Then, the difference between two costs Δ_{cost} can be written as

$$\begin{aligned}
\Delta_{cost} &= \hat{D} (1 - (|E_q| - 1) \text{Sel}(l_1) - \text{Sel}(l_2)) \\
&+ \sum_{i=2}^{|E_q|-1} \hat{D}^i (T_i - T_{i+1} - (|E_q| - 2) \prod_{j=1}^i \text{Sel}(l_j)) \\
&- (|E_q| - 1) \hat{D}^{|E_q|} \prod_{i=1}^{|E_q|} \text{Sel}(l_i).
\end{aligned}$$

Since query edges are visited in increasing order of label selectivities, we have an inequality as follows

$$\text{Sel}(l_1) \leq \text{Sel}(l_i) \leq 1.$$

With the inequality above, we have

$$i \text{Sel}(l_1)^{i-1} \leq T_i \leq i$$

Using the both inequalities above, the upper bound of Δ_{cost} can be written as

$$\begin{aligned}
\Delta_{cost} &\leq \hat{D} (1 - |E_q| \text{Sel}(l_1)) - (|E_q| - 1) \hat{D}^{|E_q|} \text{Sel}(l_1)^{|E_q|} \\
&+ \sum_{i=2}^{|E_q|-1} \hat{D}^i (i - (|E_q| + i - 1) \text{Sel}(l_1)^i).
\end{aligned}$$

Let $F_n(x)$ denote the upper bound of Δ_{cost} using x to denote $\text{Sel}(l_1)$ and n to denote the number of query edges. To show the correctness of the Lemma 2, we prove that $F_{|E_q|}(x) \leq 0$ with different number of edges when the conditions in the Lemma holds using mathematical induction.

Basis: $n = 2$: $F_2(x)$ can be written as

$$F_2(x) = \hat{D} (1 - 2x) - \hat{D}^2 x^2$$

When $x = \frac{1}{\sqrt{\hat{D}}}$, we have $F_2(x)$.

$$\hat{D} (1 - 2x) - \hat{D}^2 (\frac{1}{\sqrt{\hat{D}}})^2 = \hat{D} (-2x) < 0.$$

We also know that the derivative of $F_2(x)$ is

$$\frac{\partial F_2}{\partial x} = -2\hat{D} - 2\hat{D}^2 x < 0.$$

Combining two facts above, we know that $F_2(x) < 0$ when $\text{Sel}(l_1) > \frac{1}{\sqrt{\hat{D}}}$.

Induction hypothesis: Assume the Lemma holds when

the query has k edges.

$$F_k(x) = \hat{D}(1 - kx) - (k-1)\hat{D}^k x^k + \sum_{i=2}^{k-1} \hat{D}^i (i - (k+i-1)x^i) < 0$$

subject to $x > \frac{1}{\sqrt[k]{\hat{D}}}$.

Note that the derivative of $F_k(x)$ is

$$\frac{\partial F_k}{\partial x} = -k - k(k-1)\hat{D}^k x^{k-1} - \sum_{i=2}^k i(k+i-1)x^{i-1} < 0.$$

Induction: Using $F_k(x)$ to substitute some terms in $F_{k+1}(x)$, $F_{k+1}(x)$ can be written as

$$F_{k+1}(x) = \hat{D}(1 - (k+1)x) - k\hat{D}^{k+1} x^{k+1} + \sum_{i=2}^k \hat{D}^i (i - (k+i)x^i) = F_k(x) - \hat{D}x - \sum_{i=2}^{k-1} x^i + D^k(k - (k+1)x^k) + k\hat{D}^{k+1} x^{k+1}.$$

When $x = \frac{1}{\sqrt[k+1]{\hat{D}}}$, after replacing the x with the value in the last term and combining the last two terms, $F_{k+1}(x)$ can be written as

$$F_{k+1}\left(\frac{1}{\sqrt[k+1]{\hat{D}}}\right) = F_k(x) - \hat{D}x - \sum_{i=2}^{k-1} x^i + D^k(k - (k+1)x^k) + k\hat{D}^{k+1}\left(\frac{1}{\sqrt[k+1]{\hat{D}}}\right)^{k+1} = F_k(x) - \hat{D}x - \sum_{i=2}^{k-1} x^i - (k+1)D^k x^k.$$

Since $x = \frac{1}{\sqrt[k+1]{\hat{D}}} > \frac{1}{\sqrt[k]{\hat{D}}}$, $F_k(x) < 0$ and the rest of terms are also negative, we have

$$F_{k+1}\left(\frac{1}{\sqrt[k+1]{\hat{D}}}\right) < 0.$$

We also know that the derivative of $F_{k+1}(x)$ is negative.

$$\frac{\partial F_{k+1}}{\partial x} = \frac{\partial F_k}{\partial x} - \hat{D} - \sum_{i=2}^{k-1} ix^{i-1} - k(k+1)\hat{D}^k x^{k-1} < 0.$$

Combining two facts above, we know that $F_{k+1}(x) < 0$ when $Sel(l_1) > \frac{1}{\sqrt[k+1]{\hat{D}}}$. \square

B. QUERY SET

The query set used to compare with exemplar queries in Section 6.4 with format “<subject> <predicate> <object>” as follows:

1. D influenced Swift;
Scala influenced Swift;
Ruby influenced Swift;
Rust influenced Swift;
Swift languages Function programming;
Swift languages Procedural programming;
Swift languages Generic programming;
Swift subject_of Treehouse.
2. Lloyd Wright structures_designed Oasis Hotel;
Lloyd Wright place_of_death Santa Monica;

Lloyd Wright architectural_style Modern architecture;
Lloyd Wright influenced_by Frederick Law Olmsted;
Lloyd Wright influenced_by Frank Lloyd Wright.

3. National Audubon Society notable_types Nonprofit organization;
National Audubon Society program_partnership_s Appalachian Mountains Joint Venture;
National Audubon Society program_partnership_s Virginia Bird Conservation Initiative;
National Audubon Society named_after John James Audubon.
4. Pythagoras namesakes Pythagoras;
Pythagoras influenced Nikohl Vandel;
Pythagoras children Damo;
Pythagoras influenced Plato;
Pythagoras influenced JĀĀbir ibn HayyĀĀn.
5. Frederick County contains Ole Orchard Estates;
Frederick County events Second Battle of Winchester;
Frederick County partially_contains North Mountain;
Frederick County contains Echo Village;
Frederick County people_born_here James Brenton (1740ĀĀ\$1782);
Frederick County contains Green Acres;
Frederick County contains US Census 2000 Tract 51069050100.
6. Research subject_of Carnegie Moscow Center;
Research works Hot talk, cold science;
Research works Person or Persons Unknown;
Research organizations_of_this_type Stanford University School of Medicine;
Research schools_of_this_kind Indian Institute of Forest Management;
Research organizations_of_this_type Stanford Radiology.
7. Valve Corporation games_developed Half-Life 2;
Valve Corporation games_published Wolfenstein 3D;
Valve Corporation games_published The Maw;
Valve Corporation games_developed CS Online;
Valve Corporation games_published Half-Life 2;
Valve Corporation is_reviewed Place founded;
Valve Corporation games_published CS Online.
8. Scheme influenced Haskell;
Scheme influenced Clojure;
Scheme influenced LFE;
Scheme influenced Dylan;
Scheme influenced_by Lisp;
Scheme parent_language Lisp.
9. Cruze Control genre Southern hip hop;
Cruze Control notable_types Musical Album;
Southern hip hop artists Triple C's;
Cruze Control featured_artists Baby;
Cruze Control featured_artists Pitbull;
Southern hip hop albums Cruze Control.
10. Luke Carroll place_of_birth Sydney;
Marcus Einfeld place_of_birth Sydney;
Martin Lynes place_of_birth Sydney;
George Tarr place_of_birth Sydney;
Adventures by Disney Australia Vacation travel_destinations Sydney;
Dayne Hudson place_of_birth Sydney.